第2章

車載品質の画像処理を実現するマルチコアLSIとそのソフトウェア開発

——128個のCPUコアによる並列処理で低発熱・高性能を実現する

青山 徹

現在，セーフティ系システムの自動車への搭載が進んでいる．安全性の向上のため，センサによって障害物の有無を検出するだけでなく，障害物が何であるのか，どのような状況にあるのかなどを認識することが要求されている．そこで重要となるのがカメラから取得したデータの画像処理技術である．ここでは，NECが開発した画像処理LSIの概要とそのソフトウェア開発について解説する．このLSIは制御用プロセッサのほかに，画像処理用として128個の8ビットCPUコアを搭載している．これらのCPUコアが並列動作することで，画素ごとではなくラインごとに処理を行えるため，低い動作クロックで求められるリアルタイム性（性能）を実現できるという．（編集部）

日本における交通事故による死者の数は，近年減少傾向にありますが，平成14年（2002年）には年間8,362人もの人が交通事故で亡くなっています．これを受け，平成15年（2003年）に，小泉純一郎内閣総理大臣が談話を発表し，今後10年間で交通事故による死者数を半減注1させるという目標を掲げました(3)．ちなみに，警察庁が発表した統計資料によると，交通事故による死者数は平成15年が7,702人，平成16年（2004年）が7,358人，平成17年（2005年）が6,871人となっています(4)．

欧米でも同じような傾向が見られます．例えば，欧州では2000年の交通事故による死者が約40,000人でした．これを受け，欧州交通政策白書の中で2010年には死亡者数を20,000人にまで減らすという目標を掲げました(5)．

注1：具体的な政府目標は，「10年間で（2012年までに）死者数5,000人以下を達成」というものである．

こうした目標を達成するためには，政府（国）や自動車業界，エレクトロニクス業界などが一体となり，それぞれの立場から協力し合うことが必要です．

1 自動車における画像処理の必要性

自動車そのものに焦点を当ててみましょう．自動車のセーフティ（安全）機能は大きく二つに分類できます．

一つは，避けることのできない事故に遭遇したときに衝突エネルギーを小さくして乗員の安全を確保する，「パッシブ・セーフティ（受動的な安全性）」と呼ばれるものです．

もう一つは事故に遭遇しない（事故を未然に防ぐ）安全技術で，「アクティブ・セーフティ（能動的な安全性）」と呼ばれるものです．アクティブ・セーフティは，「走る」，「曲がる」，「止まる」という自動車の基本機能がどのような状況においても正常に動作すれば，いざというときに思いどおりに自動車を制御でき，安全を確保できるという考えかたに基づいています．

● 「何」が「どのような状態か」を認識することが重要に

最近，自動車メーカはセンシング技術を駆使して，より高度なアクティブ・セーフティ技術の研究開発や製品化を進めています．例えば，センサを用いて前方の障害物を認識し，一定の距離以下になるとブレーキ操作の支援（減速制御，運転者への警告）やシートベルトの巻き取りを行うシステムを搭載している自動車もあります．また，夜間，赤外線を照射して人や障害物を認識し，運転者に注意を促

KeyWord　パッシブ・セーフティ，アクティブ・セーフティ，発熱，リアルタイム，画像処理，IMAPCAR，RISC，1DC，separate型変数，2値化

2

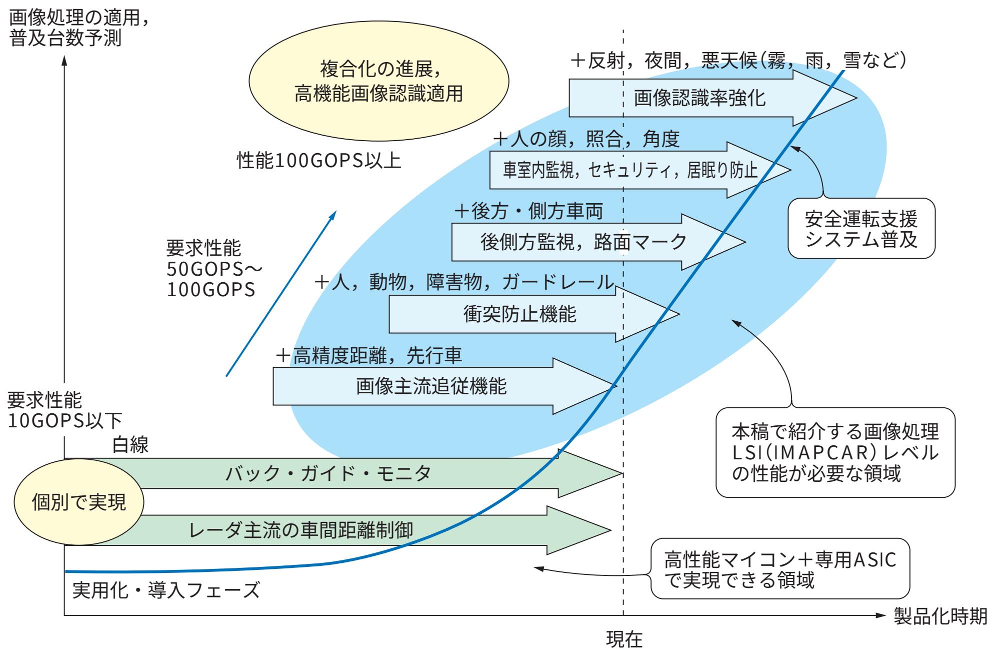


図2
画像認識適用アプリケーション
今後，運転支援システムにおいて，多種多様な画像認識処理が要求され，複合的で複雑な画像認識処理の適用が増大することが予想される．図中，各機能(矢印)の上に認識するべき対象を挙げている．各項目の頭に付いている「＋」は，「それまでの認識対象に加えて」という意味．従来はセンサ(カメラ)ごとに個別に動作して機能を実現していたが，認識対象の拡大に伴い，今後は複数のセンサが連携(複合化)して実現していくようになるだろう

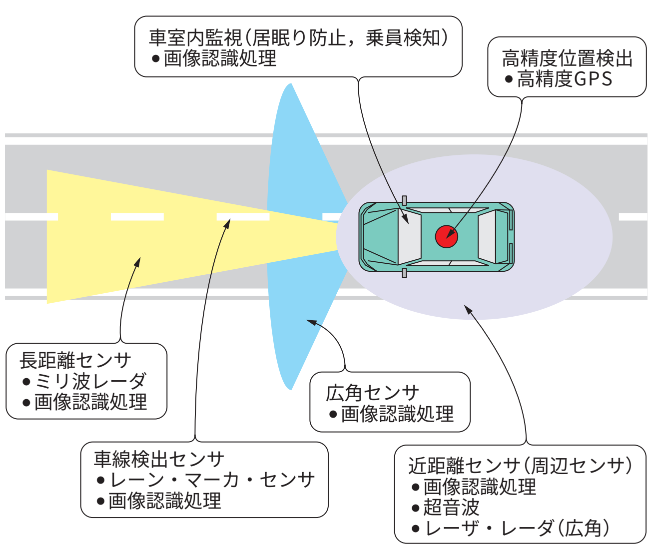


図1　安全運転支援のためのセンシング技術
走行環境におけるセンシング技術の実用化，複数のセンサの融合により，認識対象が拡大し，認識性能の向上が進む．その中でもカメラを使用した認識処理(画像認識)は，不可欠の技術となる

すシステムも実用化されています．

現在のアクティブ・セーフティ技術に使用されるセンサは，ミリ波レーダやレーザ・レーダなどを中心とした，障害物の有無を認識するシステムです．今後，自動車の安全性をさらに向上させるために，障害物は人間か車両か，どのような標識が示されているか，運転手は居眠りしていないか，信号は何色になっているかなど，より細かい情報を正確に認識することが求められるようになります．そして，人の目に代わるセンサとして，カメラを利用した画像処理技術が注目されてきています(**図1**)．

将来的には，多数のカメラを用い，それらのデータを複合して利用するため，複雑な情報をリアルタイムに処理することが求められると予想されます(**図2**)．これに伴う処理量の爆発的な増加も容易に想像できます．こうした流れに合わせて，半導体メーカは車載機器向けの画像処理LSIを開発し，複雑な情報をリアルタイムに処理する環境を提供できるように準備を進めています．

● 従来方法では性能と発熱・柔軟性がトレードオフに

アクティブ・セーフティを実現するための画像処理LSIは，複雑で膨大な画像情報をリアルタイムに，かつ正確に処理する必要があります．これに加えて，背景や照明環境，認識対象物の向きや大きさ，見えかたなど，さまざまな状況への対応が不可欠です．つまり，高い処理性能だけでなく，いろいろな状況(天候，昼夜，道路状況，走行状況など)に柔軟に対応できるプログラマビリティもあわせ持たなくてはなりません．

また，車載用半導体はそれそのものが密閉された容器に入れられ，高温の車体内に組み込まれます．そのため，信頼性を確保するためには消費電力を低減し，LSIそのものの発熱量を抑えることもたいせつです．

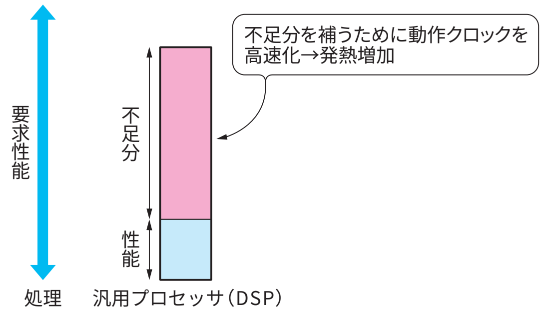


図3　従来の高速化へのアプローチ(汎用プロセッサやDSP)
高性能な汎用プロセッサまたはディジタル信号処理プロセッサ(DSP)を用いる方法では，性能不足を動作クロックを上げることで解消する．動作クロックの高速化(高性能化)に比例して発熱量が増大する．車載用途には不向き

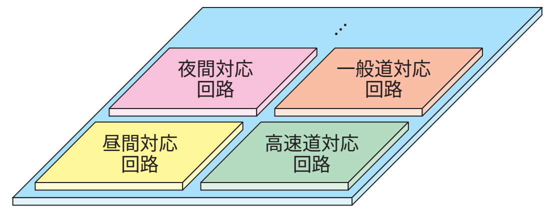


図4　従来の高速化へのアプローチ(ASIC)
専用ハードウェア(例えばASIC)を用いる方法では，状況に応じた専用回路を搭載する必要がある．回路規模が大きくなり，組み込み用途には不向き

一般的な汎用プロセッサやディジタル信号処理プロセッサ(DSP：digital signal pro-cessor)，あるいはメディア・プロセッサ注2などは，高いプログラマビリティを持ち，さまざまな状況に柔軟に対応できます．しかし，リアルタイム性が要求される複雑な動画認識処理に応用するには性能が大きく不足しています．これらのプロセッサの中には，動作クロックを極限まで上げた高性能なものもあります．この場合，高性能と引き換えに，発熱と消費電力が大きくなります．低発熱，低消費電力を目ざす車載用LSIとして適当とはいえません(**図3**)．

また別の方法として，専用ASIC(application specific integrated circuit)を開発する方法が考えられます．ハードウェアによる専用回路を作ることで，発熱や消費電力が低く，高い性能をもつLSIを実現できる可能性があります．ただし，専用ASICには開発期間の問題があります．加えて，さまざまな状況に対応するために，それぞれの状況に応じた複数の専用回路を搭載する必要があります．このことは，チップ面積(コスト)の増大を招きます．また，簡単に仕様を変更できないなどの弱点もあります(**図4**)．

上述した二つの折衷案として，DSPコア＋専用回路で実現するという解などもあると思いますが，両方の得意とする分野を持つ反面，両方の弱点もあわせ持つことになります(**図5**)．

注2：ここでは，オーディオ，ビデオ，データ，ネットワーク・プロトコルなどを処理する複数のチップの機能を統合し，1チップに集積したマルチプロセッサの総称．
注3：IMAPCARは車載用途をおもな市場としているが，例えば監視カメラの画像処理やロボットの目の代わりとなる画像処理などにも応用できる．
注4：IMAPCARは，128個のRISCコアが同時に同じ命令を実行するSIMD(single instruction multiple data)制御方式を採用している．

2　車載用途向け画像処理LSIを開発

高性能，低発熱，低消費電力，そして高い柔軟性(プログラマビリティ)を同時に実現するために筆者ら(NEC)は，これら従来のアプローチとは別の方法を用いて，車載用途の画像処理LSI「IMAPCAR」を開発しました注3．

● 128個のプロセッサによる並列処理で要求仕様を実現

本画像処理LSIは，128個のVLIW(very long instruction words)型8ビットRISCコアを1チップに集積しています．各RISCコアは100MHzという低クロックで動作し，128個のコアは並列動作(SIMD制御方式注4)します．

従来の画像処理は，単一のプロセッサで画像上の1画素を順番に処理します(**図6**)．一般には，原点(左上画素)から1ライン目を左から右方向へ順番に処理し，右端まで処理し終えたら2ライン目を処理するという流れで，最終ラ

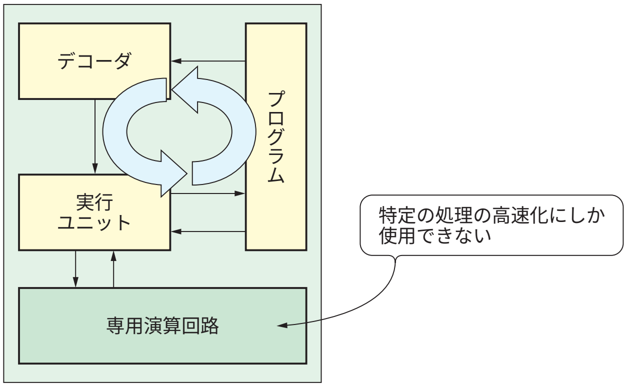


図5　従来の高速化へのアプローチ(DSPコア＋専用回路)
汎用CPU(DSP)の一部を専用回路化するという方法もある．いいとこ取りの折衷案であるが，専用回路にした特定処理でしか高速化は望めない．専用回路部分を多くすれば，ハードウェアによる専用回路(ASIC)と変わらなくなってしまう

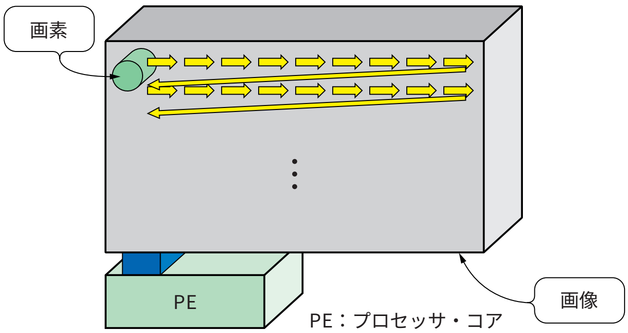


図6　単一プロセッサによる画像処理
単一の高性能プロセッサでは，画像上の画素を順番に高速に処理していく．これを「点型」処理と呼ぶ

インの右端まで処理していきます．この処理方法は，1点（1画素）ずつ処理していくので「点型」と呼ばれます．しかしこの方法では，処理能力を上げるために動作クロックを高速にすると，車載品質基準をクリアできないという問題があります．

本画像処理LSIは，128個のプロセッサ・コア（同社ではこれをPEと呼ぶ）ごとに独立したメモリ（2Kバイト）を備えています．このようなアーキテクチャをとることで，128個のプロセッサ・コアが連携して，画像の2次元面上（プロセッサ・コアごとに独立したメモリ上）でライン状につながった一連の画素を同時に処理できます．このような画像の並列処理手法を「ライン型」と呼びます．ライン型は，さらに「水平型」，「折り返し型」，「傾斜型」，「自立型」に分類できます（**図7**）．分類したそれぞれの手法を活用する典型的な画像処理には，以下のようなものがあります．

- 水平型：各種フィルタ，テンプレート・マッチング，オプティカル・フローなど
- 折り返し型：回転，FFT（高速フーリエ変換），ヒストグラム，変形など
- 傾斜型：ディザ化など
- 自立型：ラベリングなど

ライン型は，簡単に言えば，同一クロックの単一プロセッサの128倍の処理能力があります．

もちろん性能はクロック周波数だけではなく，クロックごとに発行できる命令数，メモリ容量，バスのバンド幅，分岐遅延など，いろいろな要素が複雑に絡み合って決定されます．したがって，それらを無視し，動作クロックとプロセッサ数だけを比較して「性能が128倍」というのは少し乱暴かもしれません．ここでは，大ざっぱにそれくらい高速に処理できる可能性がある，ととらえてください．

● 100MHz動作で2.4GHz汎用CPUの21倍の性能

ここでは具体的な数値により，本画像処理LSIの実力を測ってみます．ただし，最新のIMAPCARを用いた計測値がないため，ここでは1世代前のIMAP-CE[注5]というLSIの計測値を紹介します（IMAPCARは2006年中に製品化を予定している）．IMAP-CEとIMAPCARの内部の基本構成はまったく同じです．IMAPCARは，IMAP-CEの回路をさらに最適化し，高速化のための命令をいくつか追加したものなので，IMAP-CEの数値より遅くなることは考えられません．

いくつかの画像処理フィルタを用いて，ベンチマーク・テスト（処理時間計測）を行いました（**図8**）．またベンチマーク対象はIMAP-CEと2.4GHz動作の汎用CPUで，入力

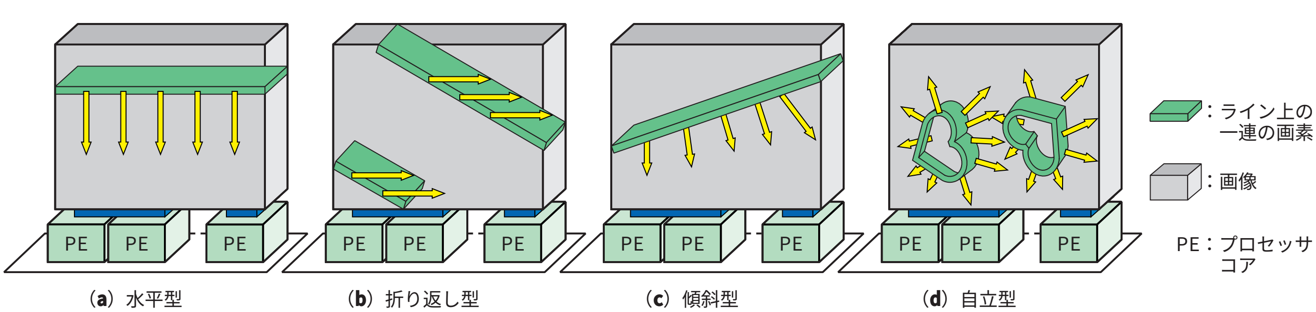


図7　IMAPCARの並列処理による画像処理
IMAPCARでは，128個のプロセッサ・コア（PE：processor element）が連携し，画像の2次元画面上でライン状につながった一連の画素を同時に処理する．これを「ライン型」と呼ぶ

注5：NECのプレス・リリース（http://www.nec.co.jp/press/ja/0302/1001.html）を参照のこと．

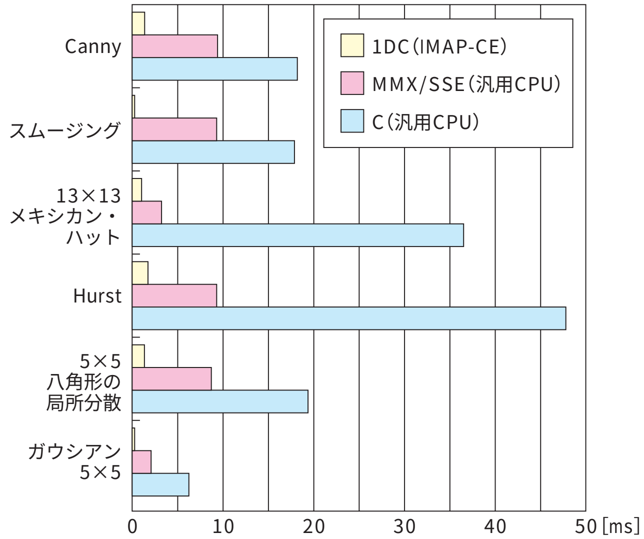


図8　IMAP-CEと汎用CPUの性能比較

画像処理で一般的に使用される画像処理フィルタを題材に，1DC，MMX/SSE，Cでプログラムを作成して処理時間の計測を行った．ベンチマーク・テストに使った画像処理フィルタは，グラフの上からCannyフィルタ(エッジ検出)，スムージング・フィルタ(平滑化)，13×13メキシカン・ハット・フィルタ，Hurstフィルタ(エッジ検出)，5×5八角形局所分散値に基づくエッジ・フィルタ，ガウシアン型重みの5×5平均値フィルタ

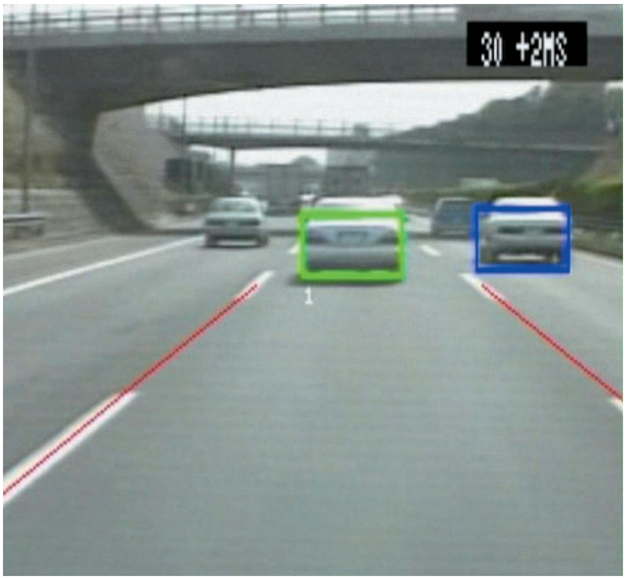


図9　IMAPCARによる先行車両検出の例

いろいろな展示会で行ったIMAPCARのデモンストレーションの一場面．先行車両認識と白線認識を行っている．処理時間は約30msで，毎フレーム処理(リアルタイム処理)している

画像は256×240画素の濃淡画像です．1DC(Cの拡張言語．詳細は後述)で記述したものをIMAP-CEで実行したとき，C言語で記述したものを汎用CPUで実行したとき，そしてMMX/SSE命令[注6]を効率的に使用したアセンブリ・コードを汎用CPUで実行したときのそれぞれの処理時間を計測しました．結果は**図8**のとおりです．

100MHz動作のIMAP-CEは，2.4GHz動作の汎用CPUと比べて，MMX/SSEコードの場合に2～13倍(平均7倍)，C記述の場合に6～21倍(平均12倍)高速であるという結果を得ることができました．これは，画像処理のごく一部分のフィルタについてのみの結果で，プログラムの作りかたも一例でしかありません．よって，大規模な画像処理アプリケーションの場合，プログラミングの方法やそのほかの要因が関係するため，そのまま当てはまる数値ではないかもしれません．しかし，動作クロック(100MHzと2.4GHz)を比べてもIMAP-CEが十分高速であることはわかると思います．

図9は，IMAPCARを使った画像認識アプリケーションの一例で，先行車両検出と白線検出を行っています．処理時間は約30msです(**図9**の右上の表示を参照)．

3 並列動作を意識したソフトウェア開発

IMAPCARの特徴である，128個のプロセッサ・コアによる並列処理を高級言語で記述できるように，C言語に必要最低限の仕様拡張を施した「1DC(One Dimensional C)」という拡張C言語を新たに開発しました．

IMAPCARのソフトウェア開発環境は，1DC言語からVLIWコードを生成するコンパイラや直感的に画像処理プログラムをデバッグするための各種ツールを統合したGUI付き1DC用ソース・デバッガ，ビデオ割り込み処理を含む標準関数ライブラリ，数多くのサンプルとドキュメントから構成されています(**図10**)．

● C言語から1DC言語への拡張

C言語の拡張部分は，大きく以下の3種類に分けることができます．

- 各プロセッサ・コア(PE)にそれぞれ実体が存在する変数を定義するための新しいデータ型：separate型(sep

注6：米国Intel社のマイクロプロセッサであるPentium，Pentium II(MMX命令)，PentiumⅢ以降(SSE命令)に搭載されているマルチメディア系の拡張命令セット．浮動小数点演算用レジスタを有効に活用し，同時に複数のデータを処理することにより，音声・画像関係の処理をCPUのみで高速に実行できる．

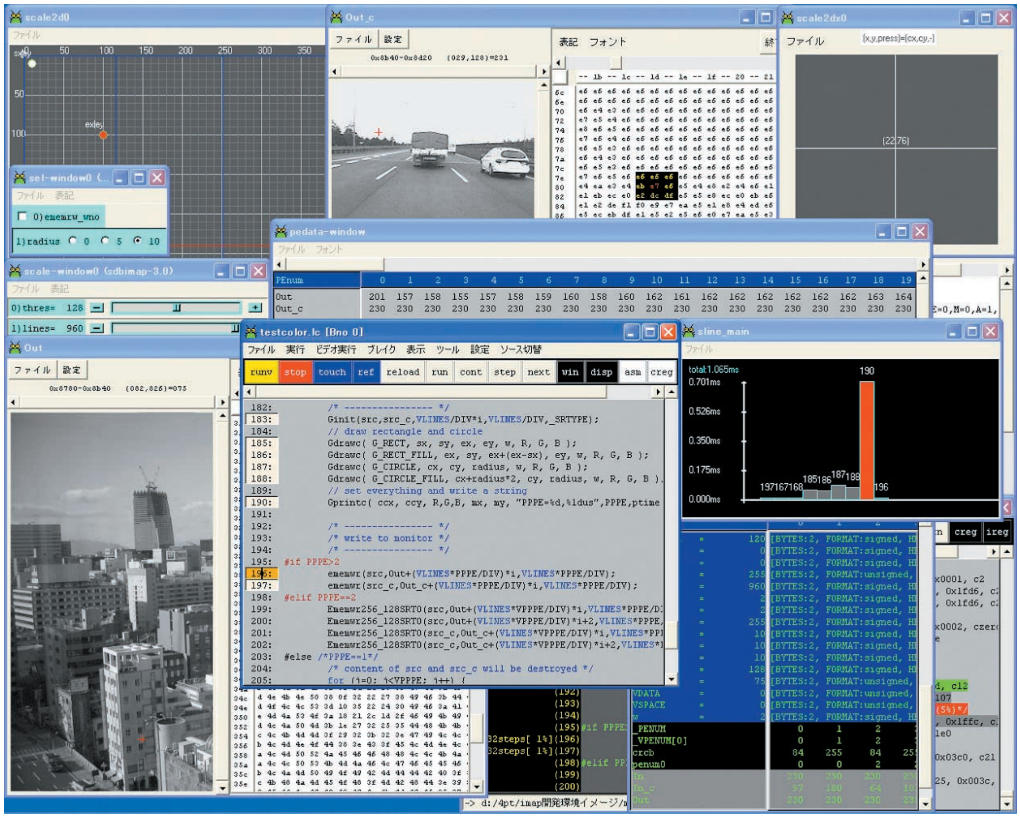
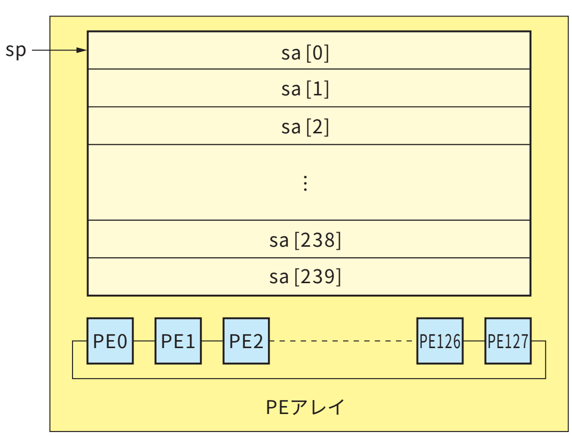


図10
開発ツール
IMAPCAR用のGUIデバッガ(sdbimap)の画面例．画像処理プログラムのデバッグやチューニングを効率良く行うための各種ツールが統合されている

```
sep unsigned char sa[240];
                    // 240行分の1バイト画素，各行は128画素幅
sep unsigned char* sp;              // sep型へのポインタ

sp = sa;
```



PE0～PE127：各プロセッサ・コア

図11　separate型変数の配置イメージ
separate型は，IMAPCARの並列処理を行ううえでもっとも重要なデータ型である．separate型変数の動きをイメージできるようになることが，高速なプログラムを作成する第一歩となる

と省略可能）

- sep型の変数を操作するための新しい演算子：:>, :<, :||, :&&, :[, :], :(, :), :=
- 並列処理を行うための新しい制御文：mif…melse, mifa…melsea, mwhile, mfor, mdo

以下に，それぞれの拡張部分を簡単に説明します．

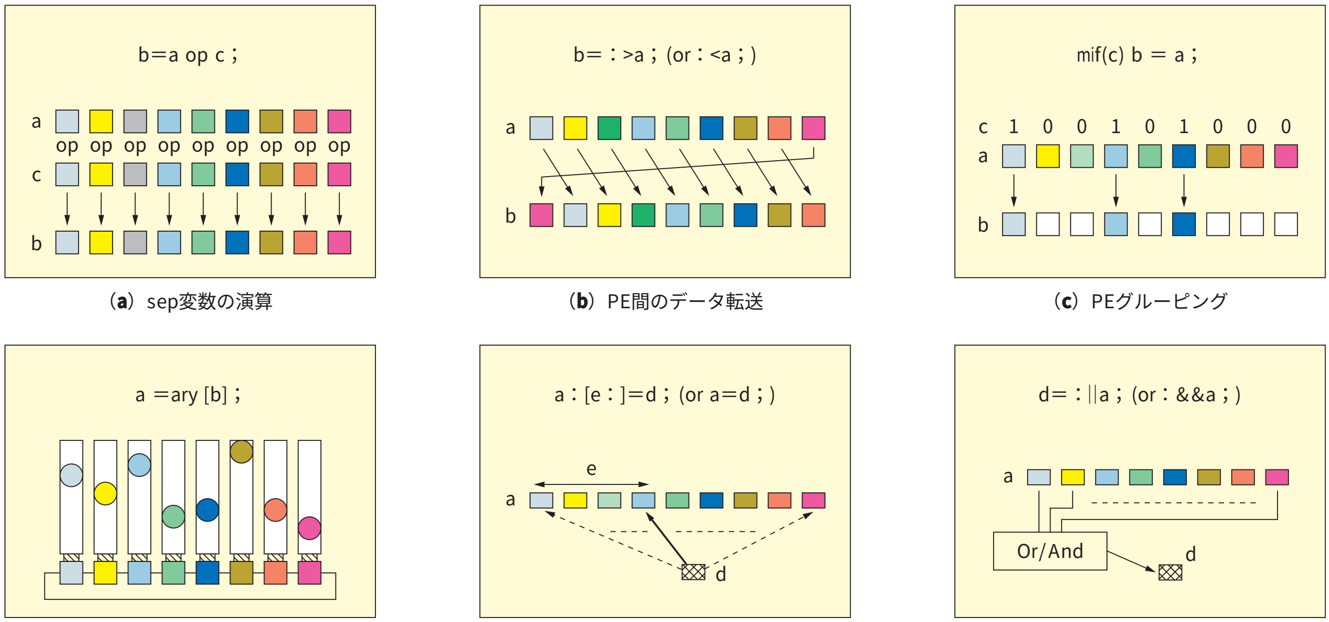
1）separate型変数

separate属性を持つ変数を宣言するには，変数宣言時にseparate修飾語を変数に先立って付けるようにします．通常，128個のプロセッサ・コア（PEアレイ）で処理を任せるデータは，このseparate型変数になります．つまり，1DCで並列アルゴリズムを扱うプログラムを記述する場合の主役です．separateは省略形sepと記述することもできます．

IMAPCARの場合，PEアレイには128個のPEが並列に配置されています．よって，以下のようにsaを宣言すると，プロセッサ・コアそれぞれのメモリ上にchar型の変数が確保されることになります．

```
sep char sa;
```

```
int       d, e;         ←制御用プロセッサ上で演算
sep char a, b;          ←PEアレイ上で演算
sep char c, ary[256]    ←PEアレイ上で演算
```



(a) sep変数の演算　(b) PE間のデータ転送　(c) PEグルーピング

(d) インデックス・アドレッシング　(e) 特定PEへのデータ設定または取り出し　(f) 全PEの論理和または論理積

図12　6種類の拡張演算記述

C言語にseparate型変数を扱うための命令を追加したものが1DC言語である．IMAPCARでは，これらの追加命令を効率良く使って，プログラミングを行う

separate型は，配列も宣言できます．**図11**は，separate型の配列イメージとポインタ・イメージです．単純なseparate型の場合，各プロセッサ・コアに同じサイズのメモリが1個ずつ確保（1ライン分）されますが，配列の場合はそれが縦に積まれるイメージになります．**図11**の場合，PEアレイ上に240ライン分のメモリが確保されます．また，separate型にもポインタ演算子が使えます．**図11**は，ポインタ変数spに配列saの先頭アドレス（&sa[0]）を代入する例です．C言語と同じように記述できることがわかります．

2）PE間データ転送演算子

プロセッサ・コア間のデータをシフト（移動）するときは，:>や:<という演算子を使用します．プロセッサ・コア上のデータをビット列に見立てると，C言語におけるビット列をシフトする場合の<<や>>という演算子に似ています．ただし，C言語のビット・シフト演算子の場合，シフトによって空いたビットには'0'が入ります．一方，1DCでは，右シフト:<の場合はあふれた先頭のデータが空いたプロセッサ・コアのデータとなります（**図12（b）**）．つまり，128回シフト（b=a:<128あるいはb=a:>128）すると，シフト前のデータと同じになります．

3）PE選択演算子

separate型データ上の任意の場所のデータに対する参照では，PE選択演算子と呼ばれる:[と:]を使用します（**図12（e）**）．例えばseparate型変数cの100番目のプロセッサ・コアのデータにアクセスしたい場合，c:[100:]と記述します．こうすることで，100番目のプロセッサ・コアのデータを取り出したり設定したりすることができます．

しかし，この演算子は特定のプロセッサ・コアの番号を指定するため，指定した以外のプロセッサ・コアは動いていないことになります．つまり，本画像処理LSIの最大の特徴である並列処理が生かされない（使えない）ことになります．処理速度を上げるためにはできるだけこの演算子を使用しないようなプログラミングを心がけることがたいせつです．

また，**図12（a）**に示すように，separate型変数どうしの演算も可能です．例えば，b=a+c;と記述すればseparate型変数どうしを足し算します．これにPE選択演算子を強引に使うと，

```
for(d=0; d<128; d++) {
    b:[d:] = a:[d:] + c:[d:];
}
```

となります．C言語風にseparate変数を配列のように使うことができますが，かなり不経済な書きかたであることがわかります．

4）PEデータ集計演算子

:||と:&&は，separate型データのすべてのプロセッサ・コア（PE）上の値について論理和，論理積を求めるPEデータ集計演算子です（**図12（f）**）．例えば，次のような値を持つseparate型変数a，b，cについて考えます．

- 変数a：PE番号0の値＝255，PE番号1～127の値＝0
- 変数b：PE番号0～127の値＝255
- 変数c：PE番号0～127の値＝0

この場合，:||aは真，:&&aは偽になります．同様に，:||bおよび:&&bは真，:||cおよび:&&cは偽になります．

5）PEデータ初期化演算子

separate型変数のプロセッサ・コア上の値として，それぞれに異なる定数を設定する初期化演算子が:(と:)です．上述のPE選択演算子を用いて，定数をそれぞれ設定することもできます．ただし，初期化演算子はコンパイル時に定数展開されるので，初期値の設定を行いたい場合はこの演算子を使用したほうが高性能なプログラムを記述できます．記述は，

```
separate型変数＝:(定数, 定数, …  :);
```

となります．この記述のように設定したい数値を羅列する方法ですが，同じ数値を繰り返して設定したい場合は下記のような記述も行えます．

```
separate型変数＝:(定数:繰り返し回数)
```

また，記述した定数の数がプロセッサ・コアの数に満たない（つまり128未満）場合，足りない残りの部分には‘0’が設定されます．最後に記述した定数を意図的に残りの部分に埋める場合，最後の定数の後ろにカンマ（,）を記述します．

6）PE選択分岐文

C言語のif～else文と同じように，separate型変数に対する選択分岐文としてmif～melseがあります（**図12（c）**）．separate型変数のすべてのプロセッサ・コア上の値に対して，いっせいにif文を実行します．mif～melse文は，以下のように記述します．

```
mif( 式1)   文1
melse       文2(melse文は省略可能)
```

式1の値は，かならずseparate属性を持っている必要があります．また，文1は式1の値がゼロでないプロセッサ・コア上で，文2は式1の値がゼロのプロセッサ・コア上で実行されます．

なお，mif文の式1の条件によって，実行されるプロセッサ・コア（アクティブなPE）と実行されないプロセッサ・コア（非アクティブなPE）が決まります．ここで決定されたアクティブ状態，非アクティブ状態は，文1，文2の中では，ネスト条件や関数コールにかかわらず，mif～melse文を抜けるまでずっと有効です．とくに関数コール先でもプロセッサ・コアのアクティブ，非アクティブの状態に変化がないことに注意が必要です．mif～melse文は，プロセッサ・コアの状態を決定（選択）する制御文であるともいえます．

7）PEデータ代入演算子

:=は，プロセッサ・コアの状態（アクティブ，非アクティブ）にかかわらず，すべてのプロセッサ・コア上でseparate属性の変数への代入処理を行うように指定する代入演算子です．下記のように，明示的にすべてのプロセッサ・コアへseparate型データを代入したい場合などに使用します．

```
sep int a, b;
a := b;  // 全PE上のb値を全PE上のaに代入する
```

8）PE選択分岐文

mifa～melsea文は，現在のプロセッサ・コアの状態（アクティブ，非アクティブ）にかかわらず，かならず条件式をすべてのプロセッサ・コア上で実行します．つまり，選択分岐文mif～melseのネスト内でmifa～melseaを使用した場合，mifaの条件式はmifの条件式に左右されることなく，すべてのプロセッサ・コア上で実行されます．mifa～melsea文は，mif～melse文と同じように記述します．

```
mifa( 式1 ) 文1
```

```
melsea          文2(melsea文は省略可能)
```

mifa文は，mif文よりもネストを考慮しない分だけ効率的なコードが生成されます．プロセッサ・コアがアクティブ状態か非アクティブ状態かを考慮する必要がない場合，mifa文を使用したほうが高速なコードを生成できます．

9）PE選択ループ文

C言語のwhile文と同じように，separate型変数に対するループ文としてmwhile文があります．mwhile文は，すべてのプロセッサ・コア上でいっせいにwhile文を実行します．mwhile文は，以下のように記述します．

```
mwhile( 式1 )   文1
```

文1の実行は，式1の値が真のプロセッサ・コア上でのみ実行されます．つまり，ループはすべてのプロセッサ・コア上の式1の値が偽になるまで繰り返されます．

10）PE選択ループ文

C言語のfor文と同じように，separate型変数に対す

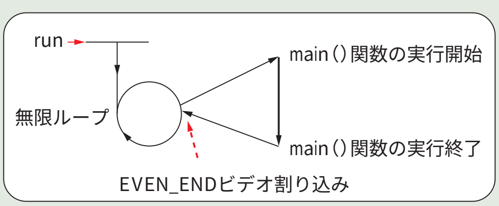
column IMAPCARのmain()関数

IMAPCARでは，main()関数の中でreturn文を使うことができます．組み込み系のプログラマの方は，main()関数の中でreturn文が使えるというと，疑問に思われるかもしれません．ソフトウェア技術者の化石のような存在の筆者もその一人でした．筆者にとってmain()関数は，スタートアップ・ルーチンの最後に呼ばれて，無限ループでぐるぐる回るものみたいなイメージがあります．IMAPCARのプログラミングを始めたころは，「main()関数でreturn？ どこへreturn？ んん？」という感じでした．

ではなぜIMAPCARでは，main()関数の中でreturn文が使えるのでしょうか？ それは，IMAPCARと一般的な組み込みマイコンでは，main()関数の位置づけ[注A-1]が少し異なるからです．これは，IMAPCARプログラムを組み始めて最初に疑問に思うことの一つでもあります．

図A-1にIMAPCARにおけるmain()関数の実行の概要(タイミング)を示します．main()関数以外に無限にループしている箇所があります．つまりIMAPCARでは，main()関数は割り込み関数の一つなのです．指定されたタイミング(図A-1ではEVEN_END[注A-2])でビデオ割り込みが発生し，main()関数が呼び出されます．main()関数では，取り込んだ1フレーム分の画像データに対する処理を実行します．したがって，main()関数でreturn文を実行しても，無限ループ部分に戻るだけでおかしな動作を行うことはありません．main()関数が割り込み関数の一つであることがわかれば，return文が書けることも納得できます．

図A-2に，main()関数の処理時間とビデオ割り込みの関係を示します．通常，main()関数は33ms間隔で呼び出されます．しかし，処理によってはその時間内にmain()関数が終了しない(図A-2のビデオ割り込み2回目)ことも考えられます．そのような場合，さらに次のビデオ割り込みが発生するまでmain()関数の起動が待たされます．main()関数の実行中に次のビデオ割り込み(多重割り込み)は発生しません．この間に入力された画像(図A-2の3フレーム目)は，処理されることなく捨てられてしまいます．

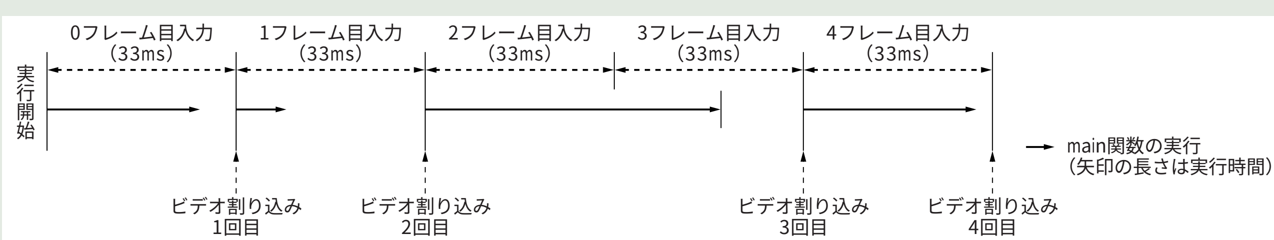


図A-1　main()関数の起動タイミング

IMAPCARのmain()関数の起動イメージ．一般的な組み込みマイコンのmain()関数とは，少し異なることに注意する．IMAPCARのmain()関数は，割り込み関数に分類される

注A-1：ここでの説明は，すべて開発ツールに含まれている標準関数ライブラリを使用した場合である．ユーザが標準関数ライブラリを使用せず，すべて自作した場合は当てはまらない．

注A-2：IMAPCARでは，main()関数実行のタイミングを以下の4種類の中から選択できる．1）プログラム開始時にmain()関数を1回実行し，プログラム実行を停止する，2）ビデオ回路のEVEN_END(偶数フィールド終了時)割り込みに同期して実行する，3）ビデオ回路のODD_END(奇数フィールド終了時)割り込みに同期して実行する，4）EVEN_END，ODD_ENDの両割り込みに同期して実行する．



図A-2　main()関数の実行時間と割り込みの関係

main()関数の処理が，次の割り込みタイミングまでに終了した場合と，終了しなかった場合の概念図．次の割り込みに間に合わない場合は，そのフレームは無視される

2

るループ文としてmfor文があります．mfor文は，すべてのプロセッサ・コア上でいっせいにfor文を実行します．mfor文は，以下のように記述します．

```
mfor(式1 ; 式2 ; 式3 ) 文1
```

文1は，式2の値が真のプロセッサ・コア上でのみ実行されます．つまりループは，すべてのプロセッサ・コア上の式2の値が偽になるまで繰り返されます．

11）PE選択ループ文

C言語のdo～while文と同じように，separate型変数に対するループ文としてmdo～mwhile文があります．mdo～mwhile文は，以下のように記述します．

```
mdo 文1 mwhile ( 式1 )
```

● 2値化の画像処理プログラムを作成する

それでは，具体的に1DCを使用した画像処理プログラムを作ってみることにします．

一般に画像処理の多くは，「入力された画像に対して，なんらかの数値的な処理を行って情報を出力すること」といえます．例えば，入力した画像に対して処理を加え，再び画像として出力することや，入力した画像からなんらかの情報を抽出して，別の結果として出力すること（文字の読み取りや認識）などです．つまり画像処理プログラムとは，それらの数値的な処理を行うためのプログラム（ソフトウェア）ということになります．

画像処理プログラムには，教科書に載っているお手本のようなものから，新しい考えかたを取り入れた斬新なものまで，多数のアルゴリズムが存在します．さらに，高速化や正確さの追求など，最適化手法の違いもあって，同じ目的の画像処理プログラムであっても，プログラマ（会社）ごとのノウハウがたくさん詰まっています．

ここでは，そのような難解なものを題材にするのではなく，画像処理の中でも代表的で，かつ比較的理解しやすい「2値化」のプログラムを作成してみます．パソコン上で動作するCプログラムと，IMAPCAR上で動作する1DCプログラムの2種類を作り，両者の違いを比較してみます．

2値化とは，濃淡画像（輝度情報）から2値（白か黒）画像を得るための数値処理を指します．一般には，しきい値処理が用いられます．しきい値をtとしたときの濃淡画像$f(x, y)$のしきい値処理は，

$$g(x, y)=\begin{cases}\text{白色} & f(x, y)\geqq t\\ \text{黒色} & f(x, y)<t\end{cases}$$

という数値処理で2値画像を得ることができます．

基準とするしきい値の値によって，処理後の画像は異なります．しきい値tは，入力画像のヒストグラム[注7]をもとに決めたり，ある特定箇所の濃淡から算出した値を利用したりとさまざまです．今回は話を単純にするため，しきい値を8ビット濃淡画像の中間値である127固定と決めてしまうことにします．言い換えれば，注目画素の輝度値が127以上であれば白色に，127未満であれば黒色に変換するプログラムを作るということになります．これをC言語風に書くと次のようになります．

```
if (注目画素輝度 < 127) {
    出力輝度 = 0;     /* 黒色 */
  }
  else {
    出力輝度 = 255;   /* 白色 */
  }
```

● C言語では画素数分ループが必要

汎用プロセッサ向けのプログラムを，C言語で作成します．汎用プロセッサでは**図6**で説明した点型処理を行うことになります．

リスト1が，2値化プログラムをC言語で書いた場合（点

リスト1　C言語による点型処理アルゴリズム

面（2次元配列）を持つ画像を処理するときの汎用プロセッサを使用したアルゴリズムの一般的な形．行の画素を順番に処理していき，行の最後で次の行の先頭へ折り返していく方法

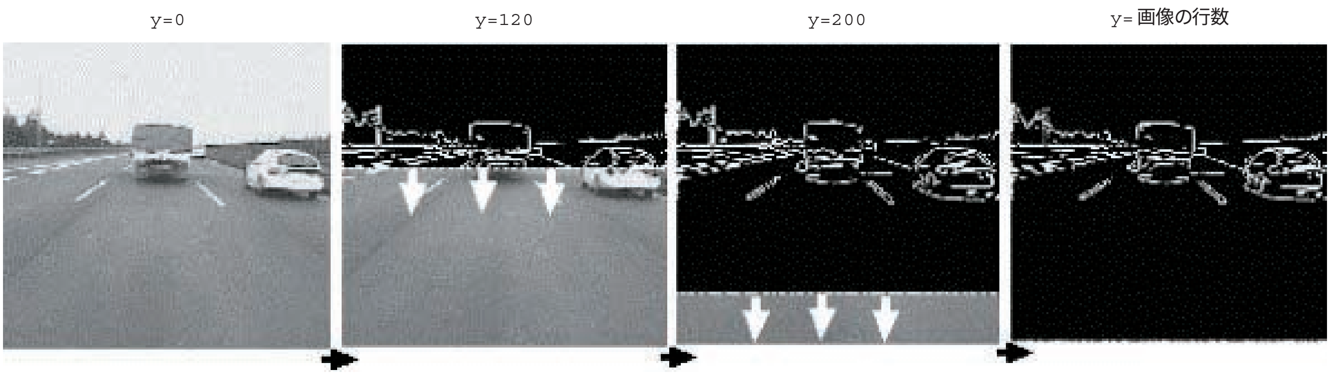
```
for(y=0 ; y<画像の行数 ; y++) {
    for(x=0 ; x<画像の列数 ; x++) {
        if (注目画素輝度 < 127) {
            出力輝度 = 0;        /* 黒色 */
        }
        else {
            出力輝度 = 255;      /* 白色 */
        }
    }
}
```

注7：ヒストグラムとは，一般的には「分布を調べる度数分布図」という意味．ディジタル画像でヒストグラムというと，画像のデータ分布を表したグラフを指す．ここでいう画像のデータ分布とは，横軸に明るさ，縦軸に明るさごとの画素数を積み上げ，画像の明暗の傾向をグラフで示したもの．

図13
1DCによるライン型処理アルゴリズム
1行（プロセッサ・コア数）を一気に処理できる1DCを使った処理アルゴリズムの概要．処理イメージのように，*Y*座標値の加算のみで1画面処理できる

```
for(y=0 ; y<画像の行数 ; y++) {
  if (注目画素輝度 < 127) {
      出力輝度 = 0;       /* 黒色 */
  }
  else {
      出力輝度 = 255;     /* 白色 */
  }
}
```

（a）プログラム（2値化処理の場合）



（b）処理のイメージ（エッジ検出フィルタの場合）

型処理アルゴリズム）の概要です．**リスト1**はごく一般的なアルゴリズムで，*X*方向のループと*Y*方向のループの二重構造になっているのがわかります．ループ回数は640×480画素の画像であれば307,200ループになり，if文判定や出力画素の設定も同回数ループ分処理されることになります．

● 横幅640画素なら*X*方向に5ループ必要

IMAPCAR向けのプログラムを，1DC言語を使用して作成してみます．IMAPCARでは，128個のプロセッサ・コアが並列に動作します．1DCを使ったプログラミングでは，並列化部分の効率の良いプログラムを，いかにうまく作るかということが重要です．この部分をうまく作るかどうかで，プログラムの性能にたいへん大きな差が生まれます．2値化プログラムはアルゴリズムそのものが単純なので，作りかたによって大きな差は生まれないと思います．しかし，より複雑なプログラムでは，わずかな変更（改良や改悪）でもすぐに数msの差として現れてきます．

IMAPCAR向けの2値化プログラムは，**図7**の水平型のように処理します．**図13**が水平型でプログラミングした場合のプログラムと処理のようすです．C言語でプログラミングしたときと比べて，二重ループがない分，if文判定や出力画素の設定の処理回数が減り，高速に処理されることが考えられます．

ただし，今のIMAPCARは128個のプロセッサ・コア（PE）しか実装していません．勘の良い読者ならすでにお気づきかと思いますが，*X*方向のループが不要なのは画像の*X*方向の幅が128画素以下の場合になります．もし横幅が640画素の場合には，*X*方向に5回分のループが必要になります．点型処理の場合，640×480画素の処理数は307,200回ですが，1DCによるライン型処理の場合は2,400回（5×480）ですみます．

それでは，実際に動作する2値化プログラムを紹介します（**リスト2**）．これは，先ほど説明した処理を640×240画素の画像に対応させ，画像の入出力処理を加えたものになります．とても簡単なプログラムですが，特徴的な点を簡単に説明しておきます．

10行目と11行目では入力画像を格納する領域と出力画像を格納する領域を，それぞれoutside属性のseparate型配列in，outという変数名で宣言しています．IMA-PCARには，大容量の外付けメモリをつなぐことができますが，ここでのoutside属性とは，その外付けメモリへの配置を意味するものです．VLINESはimapio.h内で宣言されている定数で，設定値は240になります．また，入力チャネルと出力チャネルは輝度信号用と色差信号用があるため，それぞれ2チャネル分の領域を確保しています．

また，18行目では入力画像の取り込み方式を定義しています．IMAPCARでは，「行多重方式」と呼ばれる方法と

2

リスト2　1DCによる2値化プログラム

```
#include <stddef.h>
#include <imapio.h>
#include <stdimap.h>

#define WIDTH (640 / 128)
#define THRES 127
#define BLACK 0
#define WHITE 255

outside separate unsigned char in[2][VLINES][WIDTH];  /* 入力画像格納領域 */
outside separate unsigned char out[2][VLINES][WIDTH]; /* 出力画像格納領域 */

main()
{
    separate unsigned char src[VLINES][WIDTH]; /* 作業領域 */
    int nY, nX;                                /* カウンタ */

 _SRTYPE = 1;                                  /* 行多重方式 */
    vindefOdd640( in[0], in[1] );              /* 入力画像格納領域宣言 */
    voutdefField640( out[0], out[1] );         /* 出力画像格納領域宣言 */

    ememrd( src, in[0], VLINES * WIDTH );      /* 入力画像読み込み */

    for( nY=0 ; nY<VLINES ; nY++ ) {
        for( nX=0 ; nX<WIDTH ; nX++ ) {
            mif( THRES < src[nY][nX] ) {
                src[nY][nX] = 255;             /* 輝度値を白 */
            } /* 28 */
            melse {                                 /* 29 */
                src[nY][nX] = 0;               /* 輝度値を黒 */
            }
        }
    }

    ememwr( src, out[0], VLINES * WIDTH );     /* 出力画像書き込み */
    emem2emem( in[1], out[1], VLINES * WIDTH );
}     /* 37 */
```

- かならず宣言しなければならないインクルード・ファイル群
- 扱う画像の幅の定義（640画素/128PE）
- しきい値，黒色の輝度値，白色の輝度値の設定
- 入力画像を格納する領域と，出力画像を格納する領域を，それぞれoutside属性のseparate型配列in，outという変数名で宣言
- 入力画像を画像処理するための作業領域の確保とカウンタ変数の確保
- ビデオ入力データを，10行目で宣言したinに自動的に読み込むように宣言している
- 8行目で宣言したoutから自動的にビデオ画像として出力するように宣言している
- 入力した画像（輝度データのみ）を作業領域srcにコピー
- 画像処理する行数分繰り返すためのfor文．ここでは，全行数（VLINES=240）×行多重分（WIDTH=5）だけ繰り返す
- 2値化処理部分．mif〜melse文を使用してseparate型変数を並列処理している
- 2値化処理した結果を，ビデオ出力用に確保したメモリoutへコピー
- すぐ上の行と同じく，色差データを出力用のメモリへコピーしている．ここでは，外付けメモリ間のコピーになる

「点多重方式」と呼ばれる方法で入力画像を取り込むことができます．ここでは，行多重方式を選択します．簡単に説明すると，これら二つの方法には1行分のデータ（640画素）を128画素で折り畳んだイメージ（128×5）でメモリに格納するか，5画素で折り畳んだイメージ（5×128）で格納するかの違いがあります．処理によって，扱いやすいほうで格納するように選択します．

リスト2の2値化プログラムは，実際に動くことも確認できました．並列化部分は2値化部分のみという限られた箇所だけですが，雰囲気はつかんでいただけると思います．とくに意識せずに1DCプログラムを書いても，ある程度高速な画像処理プログラムを書くことはできます．しかし，少しでも高速な処理が必要なときは，こうした並列化部分を少しでも多く，そして効率良く使わなければなりません．

参考・引用*文献

(1) 京昭倫，古賀拓也，岡崎信一郎，黒田一郎；128個の4ウェイVLIW型RISCコアを集積した車載向け動画LSI，電子情報通信学会技術研究報告，ICD2003-29，19/24，2003年．

(2) 国土交通省道路局ITSホームページ，http://www.mlit.go.jp/road/ITS/j-html/

(3)* 首相官邸ホームページ，http://www.kantei.go.jp/jp/koizumispeech/2003/01/02danwa.html

(4)* 警察庁；過去10年間（平成8年〜平成17年）の死者数の推移，http://www.npa.go.jp/toukei/koutuu1/shisha.htm

(5)* European Communities；European Transport Policy for 2010 : time to decide，2001.

(6) NECエレクトロニクスのWebサイト，http://www.necel.com/index_j.html

あおやま・とおる
NECマイクロシステム（株）

<筆者プロフィール>
青山徹．入社したころは，4ビット・マイコンのプログラムをアセンブラで作っていましたが，気がつけば128個のプロセッサ並列動作という，とんでもない怪物を扱っています．時代の流れは速いなぁとあらためて思いますが，気がつけば，自分は人生の約半分をこの業界で過ごし，すっかり「おっさん」になっていました．^^;